



ТИНЬКОФФ

Многопоточность

Основы многопоточности и примитивы синхронизации



О себе



Пришел стажером в 2020 году



Старший разработчик в Тинькофф



Развиваю партнерские сервисы Выгоды



Преподаю в Тинькофф образовании



Программа модуля



Основы работы с потоками и их устройство

Рассмотрим что такое потоки и процессы, когда,
как и зачем их использовать



Использование потоков в реальном коде

Разберем что предлагает библиотека Java для
работы с потоками



Продвинутые темы

Изучим Java memory model и посмотрим на
виртуальные потоки

План лекции



Параллелизм vs многозадачность

Разберемся зачем и когда стоит использовать потоки



Основы устройства потоков и процессов

Погрузимся внутрь потоков и посмотрим на отличие от процессов



Проблемы при работе с потоками

Задачи, которые приходится решать при работе с потоками



Примитивы синхронизации

Разберем как нам синхронизировать потоки

План лекции



Параллелизм vs многозадачность

Разберемся зачем и когда стоит использовать
ПОТОКИ

Параллельные вычисления



Подсчет суммы

12

23

87

98

187

267

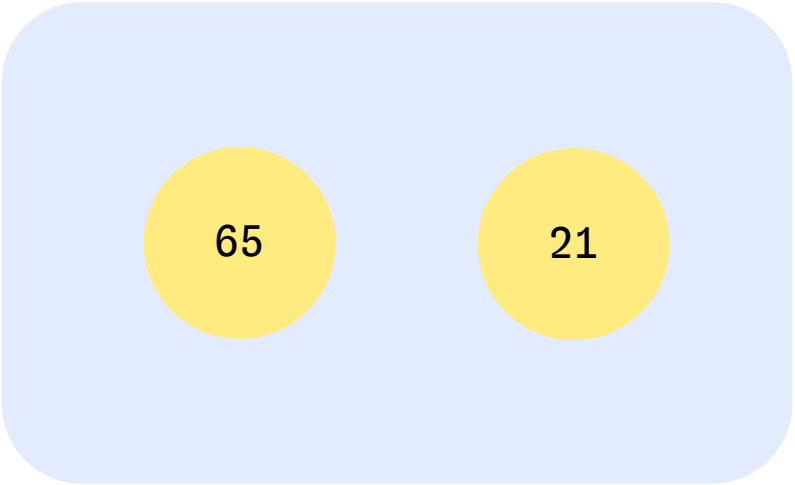
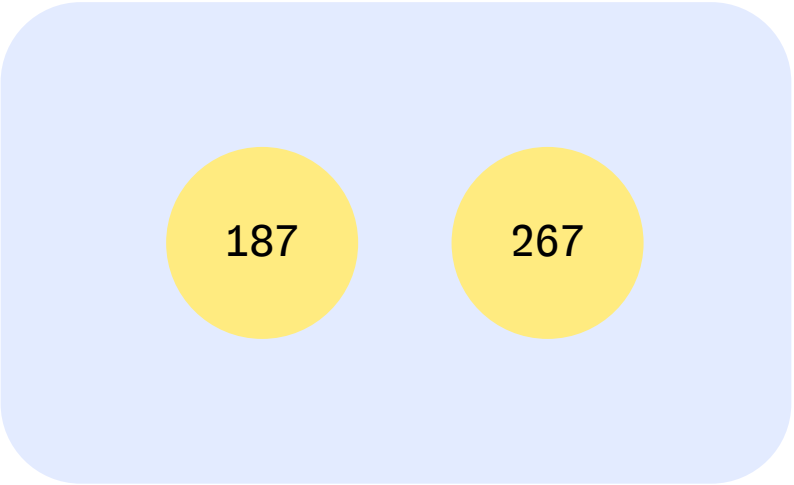
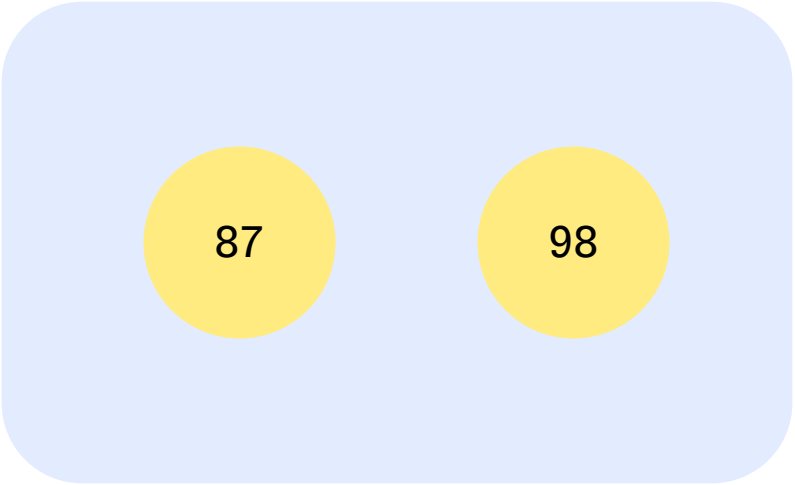
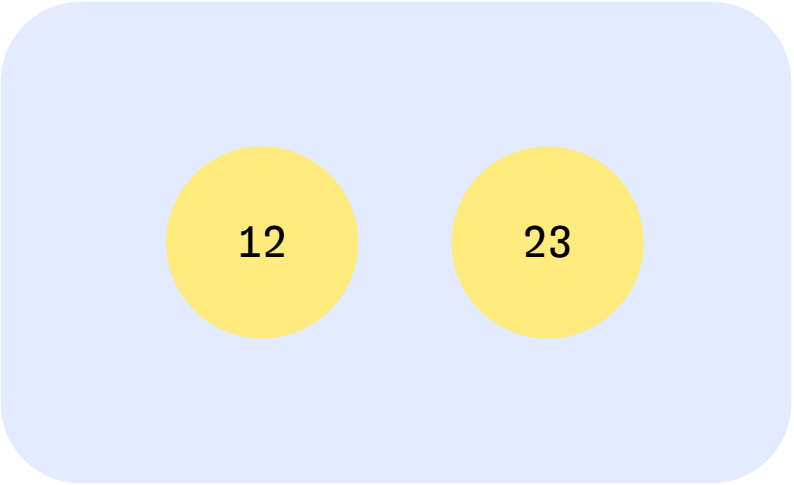
65

21

Параллельные вычисления



Разделим задачу



Многозадачность



Несколько программ одновременно

На компьютере открыто несколько программ сразу



Активна одна

Активно в один момент времени используется только одна программа.

План лекции



Параллелизм vs многозадачность


Разберемся зачем и когда стоит использовать
потоки



Основы устройства потоков и процессов

Погрузимся внутрь потоков и посмотрим на
отличие от процессов

Программы



```
String input = null;
while (!EXIT_CMD.equals(input)) {
    input = scanner.nextLine();
    String[] args = input.split(" ");
    String command = extractCommand(args);
    String result = switch (command) {
        case ADD_TASK -> commandHandler.addTask(args);
        case GET_TASK -> commandHandler.getTask(args);
        case GET_HISTORY -> commandHandler.getHistory(args);
        case GET_ALL_TASK -> commandHandler.getAllTasks(args);
        case null, default -> HELP_MESSAGE;
    };
    System.out.println(result);
}
```

Инструкции



Инструкции выполняет
CPU



Поток – сущность ОС.
Выполнение
инструкций на CPU



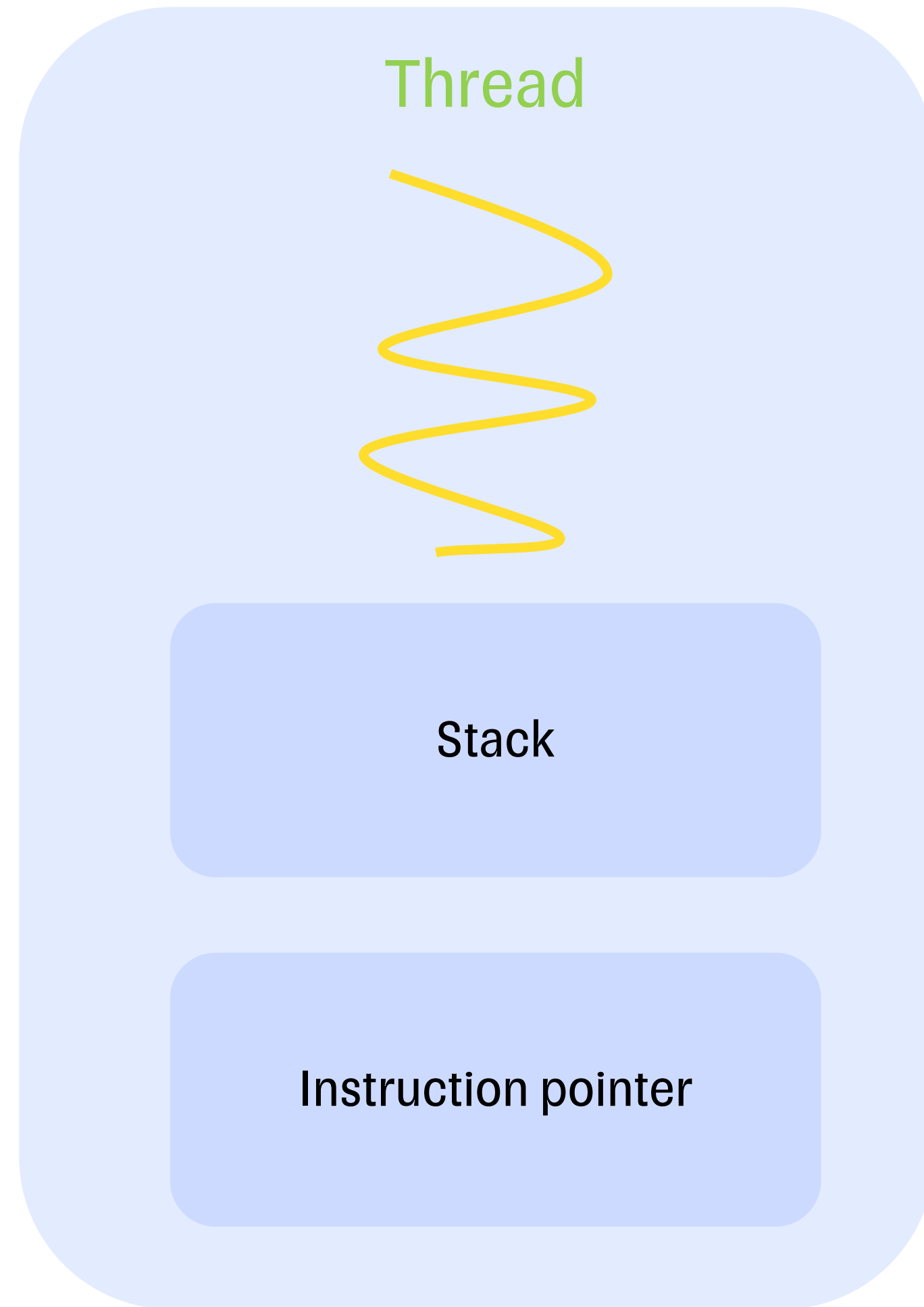
```
section .data
    number1 db 10
    number2 db 20
    result db ?

section .text
    global _start

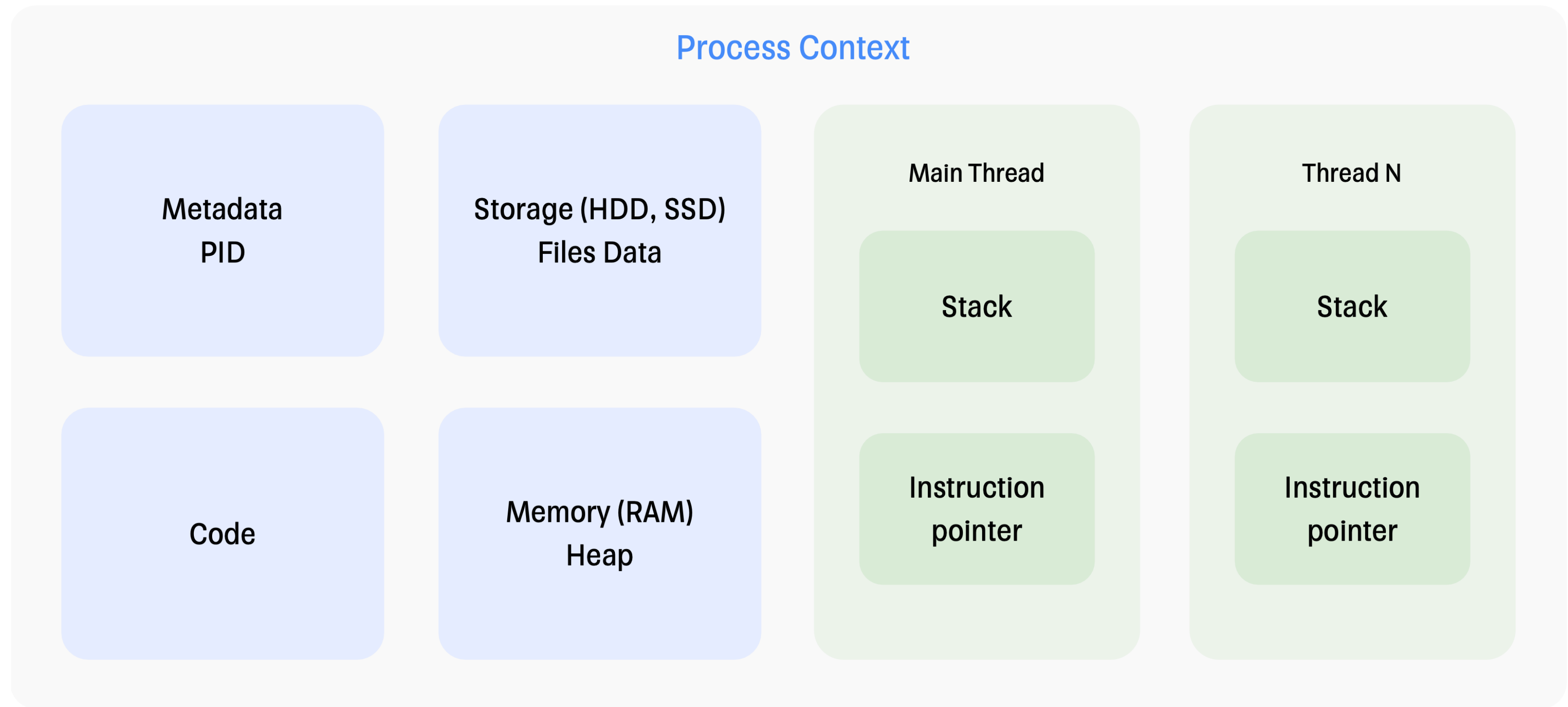
_start:
    mov al, [number1]    ; загрузить значение number1 в регистр al
    add al, [number2]    ; прибавить значение number2 к регистру al
    mov [result], al     ; сохранить результат в переменную result

    ; завершение программы
    mov eax, 1           ; номер системного вызова для exit
    xor ebx, ebx         ; код возврата 0
    int 0x80            ; выполнить системный вызов
```

Потоки



Процессы



Параллелизм

CPU 1



CPU 2



CPU 3



CPU 4



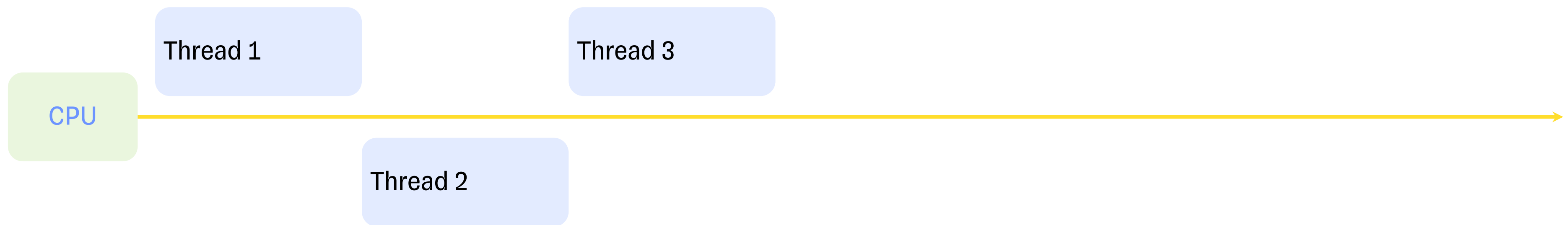
Многозадачность



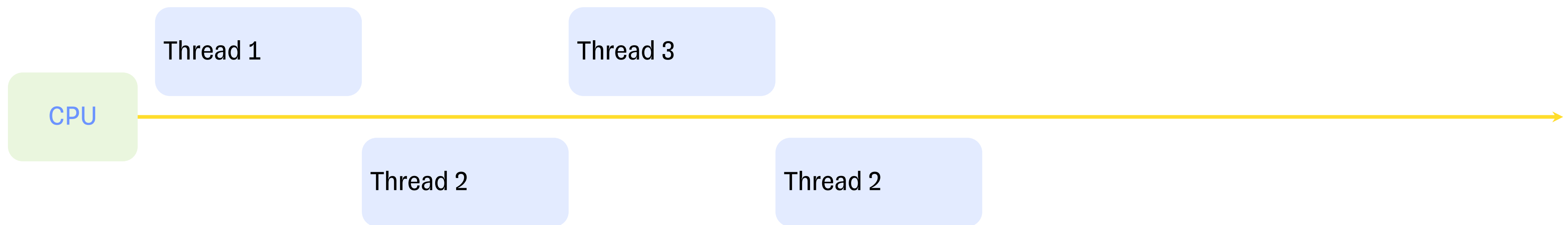
Многозадачность



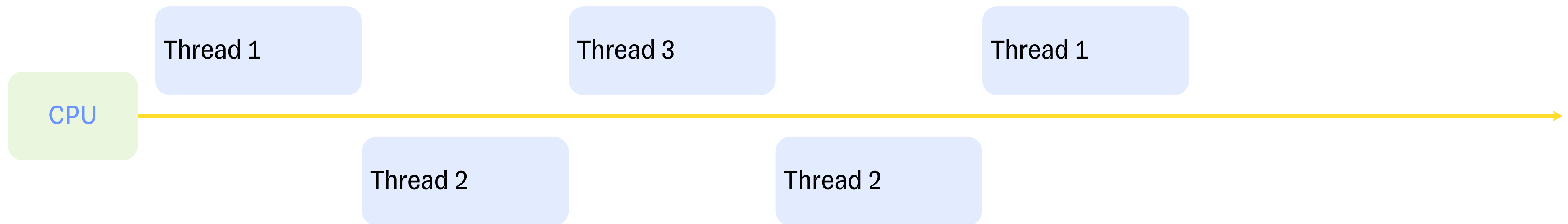
Многозадачность



Многозадачность



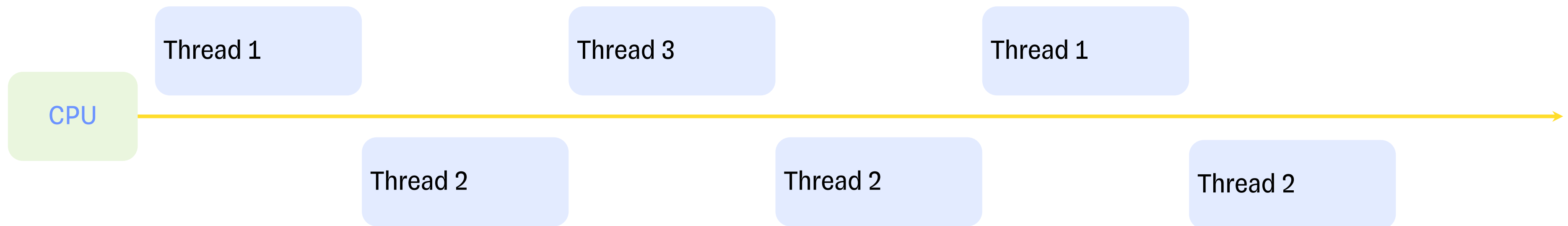
Многозадачность



Многозадачность



CPU выдает кванты времени всем потокам



Закон Амдала



Не всегда можно увеличением кол-ва ядер ускорять код



$$S_p = \frac{1}{\alpha + \frac{1 - \alpha}{p}}$$

α – доля последовательных вычислений

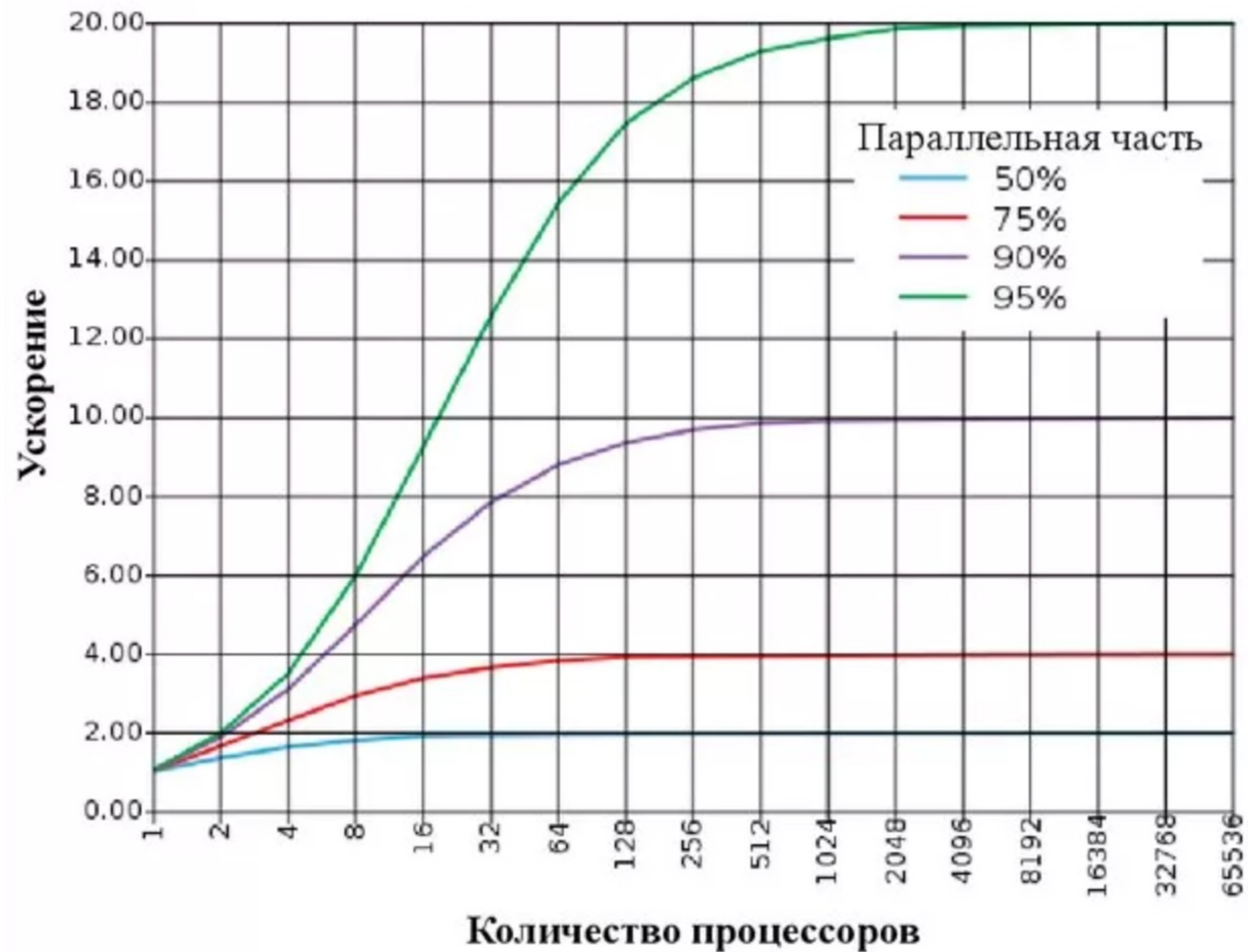
$1 - \alpha$ – доля параллельных вычислений

p – количество ядер



$$S_p = \frac{1}{\alpha + \frac{1 - \alpha}{p}} = \frac{1}{0.2 + \frac{1 - 0.2}{4}} = \frac{1}{0.4} = 2.5$$

Закон Амдала



Закон Амдала



Бесконечно ускорять код нельзя



Есть зависимость от последовательной части

План лекции



Параллелизм vs многозадачность

Разберемся зачем и когда стоит использовать потоки



Основы устройства потоков и процессов

Погрузимся внутрь потоков и посмотрим на отличие от процессов



Проблемы при работе с потоками

Задачи, которые приходится решать при работе с потоками



Примитивы синхронизации

Разберем как нам синхронизировать потоки

Потоки в Java

- Класс Thread
- start() – запускает поток
- join() – дожидается окончания потока

```
int[] numbers = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
Thread leftCalculator = new Thread(() -> {
    int sum = countSum(numbers, 0, numbers.length / 2);
    System.out.println("Left sum = " + sum);
});
Thread rightCalculator = new Thread(() -> {
    int sum = countSum(numbers, numbers.length / 2, numbers.length);
    System.out.println("Right sum = " + sum);
});
leftCalculator.start();
rightCalculator.start();
try {
    leftCalculator.join();
    rightCalculator.join();
} catch (InterruptedException e) {
    System.out.println("Error while join threads " + e.getMessage());
}
System.out.println("Finished count");
```

Потоки в Java



Можно через
наследование



Переопределяем метод
run()

```
public class Calculator extends Thread {

    private final int[] numbers;
    private final int from;
    private final int to;

    public Calculator(int[] numbers, int from, int to) {
        this.numbers = numbers;
        this.from = from;
        this.to = to;
    }

    @Override
    public void run() {
        System.out.println("sum from %d to %d = ".formatted(from, to) +
countSum(numbers, from, to));
    }

    private static int countSum(int[] numbers, int left, int right) {
        int sum = 0;
        for (int i = left; i < right; i++) {
            sum += numbers[i];
        }
        return sum;
    }
}
```

Race condition



Что выведется в результате?

```
public static class RaceCondition {
    private int value = 0;
    public void increment() { value++; }
    public void decrement() { value--; }
    public int getValue() { return value; }
}

public static void main(String[] args) {
    var raceCondition = new RaceCondition();
    var incrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.increment();
        }
    });
    var decrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.decrement();
        }
    });
    incrementor.start();
    decrementor.start();
    try {
        incrementor.join();
        decrementor.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println(raceCondition.getValue());
}
```


Race condition



Что выведется в результате?



Значения будут не
ожидаемые

```
public static class RaceCondition {
    private int value = 0;
    public void increment() { value++; }
    public void decrement() { value--; }
    public int getValue() { return value; }
}

public static void main(String[] args) {
    var raceCondition = new RaceCondition();
    var incrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.increment();
        }
    });
    var decrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.decrement();
        }
    });
    incrementor.start();
    decrementor.start();
    try {
        incrementor.join();
        decrementor.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println(raceCondition.getValue());
}
```

Sharable state

→ value – разделяемая память

→ Read и Write операции

→ Критическая секция

```
public static class RaceCondition {
    private int value = 0;
    public void increment() { value++; }
    public void decrement() { value--; }
    public int getValue() { return value; }
}

public static void main(String[] args) {
    var raceCondition = new RaceCondition();
    var incrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.increment();
        }
    });
    var decrementor = new Thread(() -> {
        for (int i = 0; i < 100_000; i++) {
            raceCondition.decrement();
        }
    });
    incrementor.start();
    decrementor.start();
    try {
        incrementor.join();
        decrementor.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println(raceCondition.getValue());
}
```

Read-Modify-Write



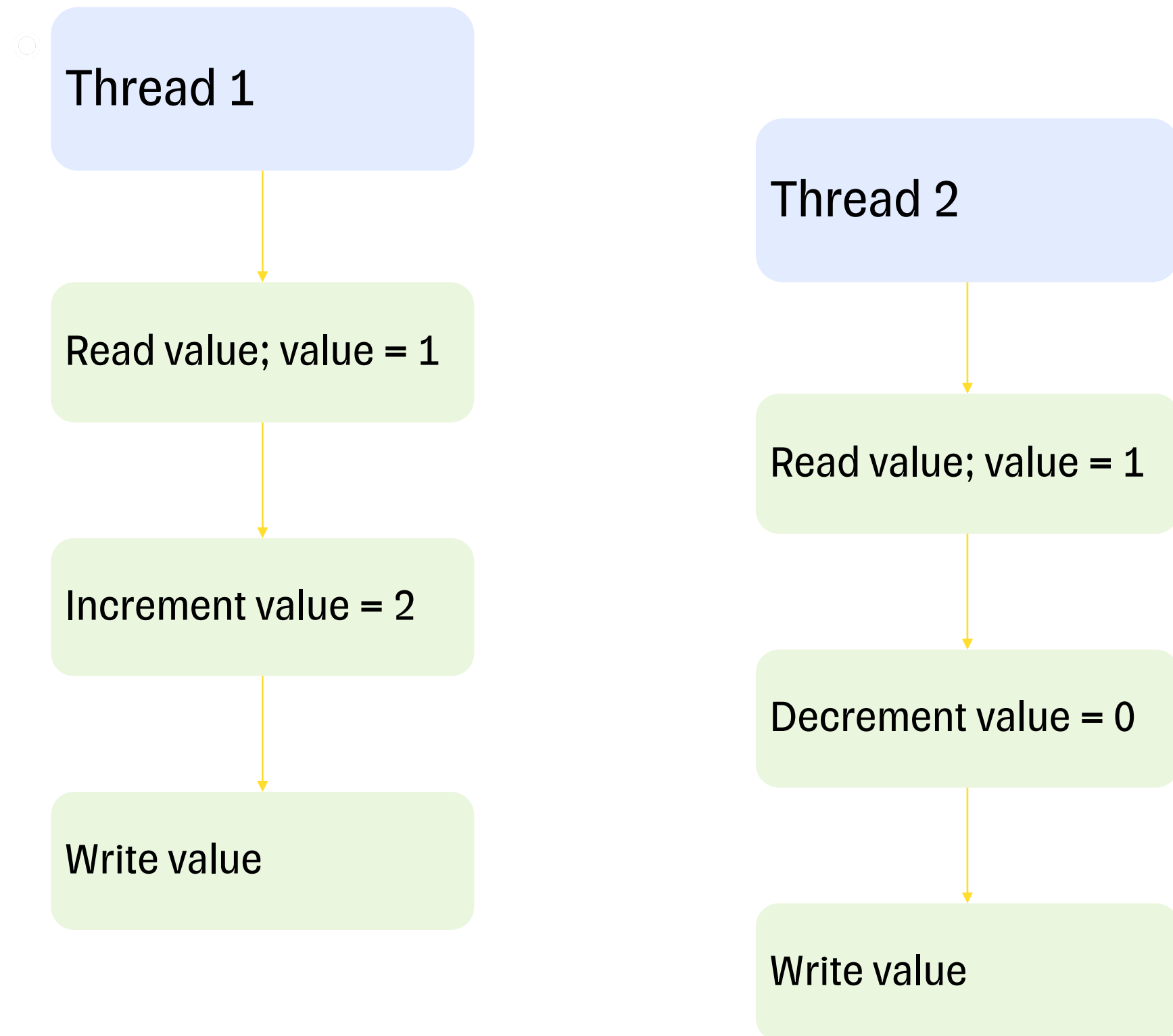
Операция инкремента
неатомарная

Read x



Modify $x = x + 1$

Write x



synchronized



block/method



Нужно делать synchronized
блоки как можно меньше



Блокировка берется на
объекте this



Можно использовать для
static методов



Одну и ту же блокировку
можно брать неоднократно

```
public static class RaceCondition {  
  
    private int value = 0;  
  
    public synchronized void increment() {  
        value++;  
    }  
  
    public synchronized void decrement() {  
        value--;  
    }  
  
    public int getValue() {  
        return value;  
    }  
}  
  
public static void main(String[] args) {  
    RaceCondition raceCondition = new RaceCondition();  
    var incrementor = new Thread(() -> {  
        for (int i = 0; i < 100_000; i++) {  
            raceCondition.increment();  
        }  
    });  
    var decrementor = new Thread(() -> {  
        for (int i = 0; i < 100_000; i++) {  
            raceCondition.decrement();  
        }  
    });  
    incrementor.start();  
    decrementor.start();  
  
    try {  
        incrementor.join();  
        decrementor.join();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
  
    System.out.println(raceCondition.getValue());  
}
```

Data visibility



Каким будет вывод?

```
static int nextCustomerNumber = 0;

public static void main(String[] args) throws Exception {
    new CustomerController(4).start();
    new Queue().start();
}

static class CustomerController extends Thread {
    private final int customerNumber;
    public CustomerController(int customerNumber) {
        this.customerNumber = customerNumber;
    }
    @Override
    public void run() {
        while (nextCustomerNumber < customerNumber) {
        }
        System.out.printf("Клиент наконец дошел %d\n", nextCustomerNumber);
    }
}

static class Queue extends Thread {
    @Override
    public void run() {
        while (nextCustomerNumber < 20) {
            System.out.printf("Вызываем клиента #%d\n", nextCustomerNumber++);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```


Data visibility



Каким будет вывод?



Изменения одного потока не видны в другом

```
static int nextCustomerNumber = 0;

public static void main(String[] args) throws Exception {
    new CustomerController(4).start();
    new Queue().start();
}

static class CustomerController extends Thread {
    private final int customerNumber;
    public CustomerController(int customerNumber) {
        this.customerNumber = customerNumber;
    }
    @Override
    public void run() {
        while (nextCustomerNumber < customerNumber) {
        }
        System.out.printf("Клиент наконец дошел %d\n", nextCustomerNumber);
    }
}

static class Queue extends Thread {
    @Override
    public void run() {
        while (nextCustomerNumber < 20) {
            System.out.printf("Вызываем клиента #%d\n", nextCustomerNumber++);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

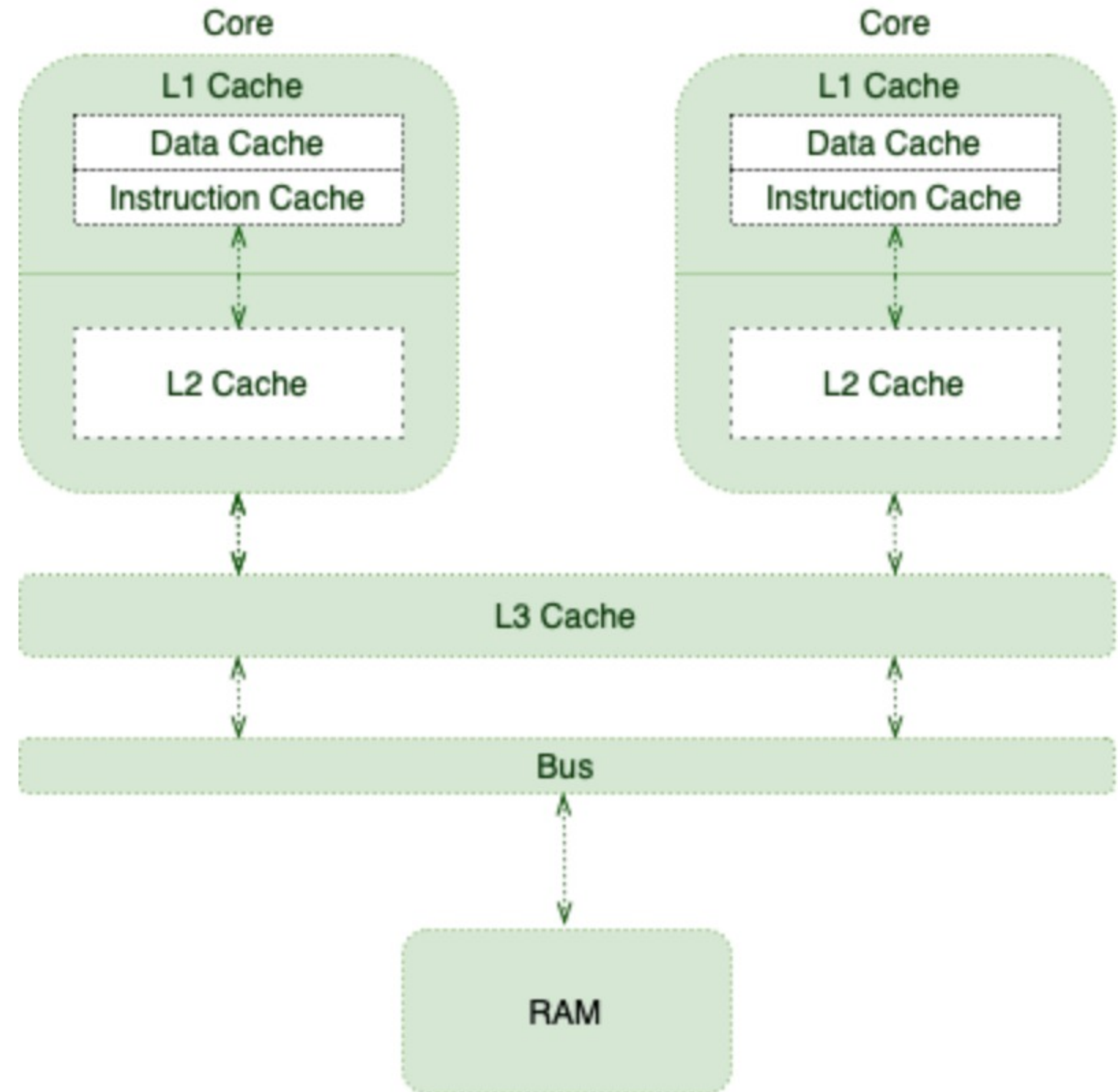
Data visibility



Из-за оптимизаций кода
данные кэшируются



Каждый поток может
работать со своей копией



Volatile



Кэши будут
синхронизированы



Все операции до работы с
volatile не будут перемешаны

```
static volatile int nextCustomerNumber = 0;

public static void main(String[] args) throws Exception {
    new CustomerController(4).start();
    new Queue().start();
}

static class CustomerController extends Thread {
    private final int customerNumber;
    public CustomerController(int customerNumber) {
        this.customerNumber = customerNumber;
    }
    @Override
    public void run() {
        while (nextCustomerNumber < customerNumber) {
            System.out.printf("Клиент наконец дошел %d\n", nextCustomerNumber);
        }
    }
}

static class Queue extends Thread {
    @Override
    public void run() {
        while (nextCustomerNumber < 20) {
            System.out.printf("Вызываем клиента #%d\n", nextCustomerNumber++);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Deadlocks



Оба потока заблокированы

```
private static class Account {
    private BigDecimal money;
    public Account(BigDecimal money) {
        this.money = money;
    }
}

public static void moveMoney(Account from, Account to, BigDecimal value){
    System.out.println("Starting moving money");
    synchronized (from) {
        sleep(1000);
        synchronized (to) {
            System.out.println("Moving money...");
            if (from.money.compareTo(value) > 0) {
                to.money = to.money.add(value);
                from.money = from.money.subtract(value);
            }
        }
    }
}

public static void main(String[] args) throws InterruptedException {
    Account account1 = new Account(BigDecimal.valueOf(4000));
    Account account2 = new Account(BigDecimal.valueOf(3000));
    new Thread(() -> moveMoney(account1, account2, BigDecimal.valueOf(100))).start();
    new Thread(() -> moveMoney(account2, account1, BigDecimal.valueOf(100))).start();
}
```


ReentrantLock



Можно использовать
ReentrantLock

```
private static class Account {
    private Lock lock = new ReentrantLock();
    private BigDecimal money;
    public Account(BigDecimal money) {
        this.money = money;
    }
}

public static void moveMoney(Account from, Account to, BigDecimal value) {
    System.out.println("Starting moving money");
    try {
        if (from.lock.tryLock(100, TimeUnit.MILLISECONDS)) {
            try {
                sleep(1000);
                if (to.lock.tryLock(100, TimeUnit.MILLISECONDS)) {
                    try {
                        System.out.println("Moving money...");
                        if (from.money.compareTo(value) > 0) {
                            to.money = to.money.add(value);
                            from.money = from.money.subtract(value);
                        }
                    } finally {
                        to.lock.unlock();
                    }
                }
            } finally {
                from.lock.unlock();
            }
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Обратная связь





ТИНЬКОФФ

Он такой один

